I. J. Computer Network and Information Security, 2012, 11, 21-31
Published Online October 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2012.11.03

Analysis and Evaluating Security of Component-Based Software Development: A Security Metrics Framework

Irshad Ahmad Mir[1], S.M.K Quadri[2]
[1]Research Scholar, Post Graduate Department of Computer Science, University of Kashmir, India,
[2]Director, Post Graduate Department of Computer Science, University of Kashmir, India
irshad.mir@hotmail.com, quadrismk@hotmail.com

Abstract — Evaluating the security of software systems is a complex problem for the research communities due to the multifaceted and complex operational environment of the system involved. Many efforts towards the secure system development methodologies like secSDLC by Microsoft have been made but the measurement scale on which the security can be measured got least success. As with a shift in the nature of software development from standalone applications to distributed environment where there are a number of potential adversaries and threats present, security has been outlined and incorporated at the architectural level of the system and so is the need to evaluate and measure the level of security achieved . In this paper we present a framework for security evaluation at the design and architectural phase of the system development. We have outlined the security objectives based on the security requirements of the system and analyzed the behavior of various software architectures styles. As the component-based development (CBD) is an important and widely used model to develop new large scale software due to various benefits like increased reuse, reduce time to market and cost. Our emphasis is on CBD and we have proposed a framework for the security evaluation of Component based software design and derived the security metrics for the main three pillars of security, confidentiality, integrity and availability based on the component composition, dependency and inter component data/information flow. The proposed framework and derived metrics are flexible enough, in way that the system developer can modify the metrics according to the situation and are applicable both at the development phases and as well as after development.

Index Terms — Security Evaluation, Software Architecture, Security metrics, Component-dependencies

I. INTRODUCTION

To incorporate the security into the software development process seems to be a challenging task and the evaluation of security is even more challenging. The main reason behind it is the inappropriateness in the specification of non-functional security requirements at hand during the early system development stages and the complex nature of operational environment. A well-established scale against which we can measure the level of security a software system exhibit is still a great challenge for the research community. Traditionally security is treated as an afterthought, in which the protection mechanisms employed after the development stages of the software [1]. To evaluate the software system for security is one of the most critical aspects of security which got attention lately. Almost 100 years ago Lord Kelvin stated “If you can’t measure it you can’t improve it. Security metrics are seen as an important factor in making sound decision about various aspects of security architecture & controls, to the effectiveness and efficiency of security operations [2]. A security evaluation framework and derived security metrics that provide an indicator of security and identify the most critical elements of software system at early development stages is required and is the focus of our current study. The evaluation of software security is a very hard problem due to the complex nature of operational environment. Varieties of software architectures and design approaches have been proposed and utilized in industries, such as object oriented analysis and design (OOAD), component based design (CBD), Enterprise architecture framework (EA), and newly the Service oriented analysis and design (SOAD). The most common approach towards the development of software system is that, a software system is treated as an integrated set of small components which share/transmit process and store the information and data. Each component provides certain level of functionality and abstraction to other accessing elements. As with the current networked cyber space various threats to software systems are present in the operational environment, security becomes the utmost important issue. Various

secure system development methodologies have been proposed such as SecSDLC and Secure System Development by Microsoft (SDL) [3]. SDL is the process that Microsoft has adopted for the development of software that needs to stand with malicious attack. In this SDL of Microsoft, they have incorporated the security related activities such as development threat models for design phase and static analysis code scanning etc. Incorporating the security in development phases is one aspect, the question arises how much we achieved or can we predict and evaluate the security at the early development phases? The answer to this is the security metrics. As far as security evaluation in general and particularly at architectural design phases of the software development is concerned, it is very hard problem to be solved and has got attention lately. Of course due to the possibility of threats that are unknown, no system will ever be 100 % secure [4] but the metrics which provide an early indicator of security is helpful and needed. According to [5] the following approach should enable new type of security metrics.

- Identify security features that require system level functions.
- Evaluate the extent to which security features protect system from deliberate damage that would cause system failure.
- Device verification and validation metric at system level that show security requirements are met.

Evaluating the credibility of the security features does not solve the problem completely as new attacks and vulnerabilities are emerging but a scale of measurement that can be applicable at both the development stages and the operational phase is the requirement of the IT industries and also is the focus of our current study. In the current study we initially focused on the specification of non-functional security requirements. We argue that the specified security requirements generally must satisfy one or more well established security attributes of fig.1. We have analyzed the various software architectural styles and their place in software development. Our main focus will be on Component based Software Development (CBD) , because all the other architectural styles follow and inherit the CBD . we believe that the component level is the best level of software architecture in a way that it neither provides the more complex fine-grained details like Object Oriented Analysis and design(OOAD) nor too coarse-grained like Service Oriented Architecture(SOA). Further we have proposed a security evaluation framework and derived the security metrics for the three main attributes of security, which are the confidentiality, integrity and availability. Our evaluation is based on the dependencies among the various distributed components and on the flow of data/information among them. The derived metrics can provide an early indicator for the developers to identify the critical components in system and act as the correction measures in making early decisions about the component composition and protection mechanism. These early indicators can reduce the further maintenance cost considerably.

The rest of the paper is organized as: Section 2 provides the overview of the security requirement specification, section 3 we analyze the software architectures and design. In section 4 we analyze the behavior of Component Based Design (CBD), its various models and composition. In section 5 we present a security evaluation framework and derive various metrics. Finally the conclusion in section 6.

II. SOFTWARE SECURITY REQUIREMENTS

The need to consider security from the ground is a fundamental step to the secure system development [6]. The requirement phase is the opportunity for the product team to consider how security be integrated into the development process. While these early plans are subject to change as the project proceeds, but early articulation of these plans helps to ensure that no requirements are overlooked. According to [7] the most current software requirements specifications are either (i). Totally silent regarding security (ii). Merely specify vague security goals (iii). Specify commonly used security mechanisms as architectural constraints. In the first case, security is not considered in development stages. In second the specified security requirements are unstructured and very hard to evaluate. Third case bind the architectural decision too early resulting in inappropriate security mechanisms.

Security requirements normally come under non-functional requirements of the system which represent how software performs rather than what it does as in functional requirements. Information security cannot be only represented by non-functional requirements, since security goals often motivate new functionality, such as monitoring, intrusion detection and access controls which in turn need functional requirements [8]. Specification of security requirements in a concrete, well established and unambiguous form is still lacking in the security engineering community which poses a difficulties is evaluating the system against these requirements. In [9] the various security requirements characteristics such as completeness, correctness, feasibility, necessity, prioritization, unambiguity and verifiability have been presented. These characteristic aims at the specification of security requirements in a structured manner.

The reason behind the difficulty in security requirement specification is the organizational specific needs and the complex nature of the operational environment under which these systems operate. These requirements are usually drawn from the higher level organizational policies. In [6] a framework for

requirement centric and model based information security evaluation has been proposed.

In our evaluation framework we considered the security requirement specification ultimately should satisfy and come under one or more of the well-established security attributes as shown in above fig.1. So at first place the system should be evaluated against the common security attributes to satisfy the security requirements.

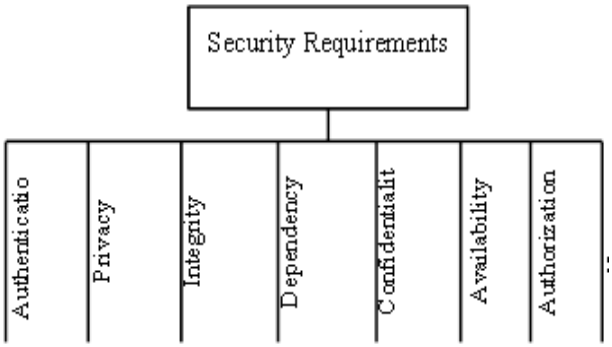

Figure 1: Security Requirement centric security attributes

III. SOFTWARE ARCHITECTURE AND DESIGN

Constructing a software system by composing prefabricated or newly developed components is always an initiative and attractive vision for software development [10]. There is a consensus on the fact that for any large software systems the overall structure, i.e. the high-level organization of computational elements and interaction between those elements is the critical aspect of the design [11]. Many definitions and concepts regarding the software architecture have been mentioned in literature. Architecture is a fundamental organization of a system embodied in its components, their relationship to each other and to environment and the principles its design and evolution [12]. These definitions of software architecture not only acknowledge structural elements but the composition of architectural elements, their relationship, the connecters needed to support their relationships, their interface and their partitions again. Unified Modeling Language (UML) diagrams are widely used in representing the functional architecture and design of the software. The UML considerably lacks in collecting the nonfunctional behavior like security issues of the system. Software engineers address the high-level aspect of the system while describing the software architecture and design, such as the overall organization, the decomposition into components, assignment of functionality into components and way the components interact. Design recognize a number of distinct design levels, each with its own design issues, model, notations , componentry and required analysis technique [13]. The main focus of architecture and design is on the functionality of the system. The security is given less attention, if given that only to the security mechanism and architectural constraints. The focus towards evaluation of the security of a given architecture and design is negligible, while these early indicators provide the significant mean of improvement and reduces the considerable cost in further patching and management. Studies have showed that it is five to hundred times cheaper to fix fundamental flaws in systems at early development stages [14], making architectural analysis more cost effective than fixing the bugs after the system is developed. Also the use of software architecture for predicting the quality attributes of overall system is one of the original motivations in the field of software architecture [15].

IV. COMPONENT-ORIENTED DESIG

Various software modeling disciplines such as Object Oriented Analysis and Design (OOAD), Enterprise Architecture Framework (EA), Service Oriented Analysis and Design (SOAD) have been presented in literature and used in the software development process. Component Based Software Engineering (CBSE) or Component Based Design (CBD) is a successor of OOAD [16] and has been supported by commercial component frameworks such as Microsoft COM, Sun's EJB, and COBRA. The OOAD, CBSE and SOA, they all differ in the way of level of abstraction in design specifications of the system. In OOAD the system is viewed as classes and objects their relationships like inheritance, polymorphism etc. In SOAD the system is specified at higher level of abstraction in which the system is generally stateless, fully encapsulated, satisfying a generic business service [17]. Adopting CBD does not mean that OOAD is useless, instead the lower level implementation of classes follow the OOAD. Similarly service uses the lower level components for its implementation. Fig.2. below shows how the technology layer can be applied to application architecture to provide the more coarse-grained implementation [18].

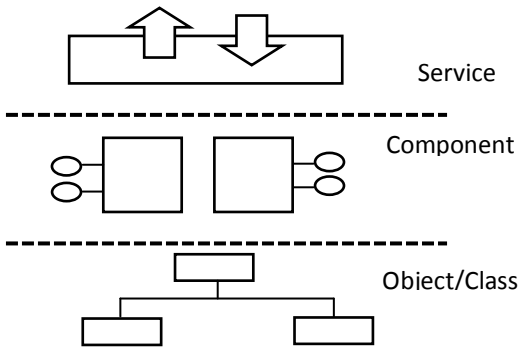

Figure 2: Different Software Architecture Layers.

The term coined to refer to this part of the system is the application edge, reflecting that a service is a great way to expose an external view of system, with internal

reuse and composition using traditional component design

As shown in fig.2 CBD is the best and manageable level of abstraction for large scale software development in a way that it is neither much coarse-grained like SOA nor much fine grained like OOAD which makes it the suitable for analyzing the quality attributes like reliability, security etc. The fundamental unit of a large scale software construction is a component. In Component based software design (CBD) the system is structured as a collection of components and their interconnection and composition. According to [19] a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. The abstract view of a component is shown in fig. 3. It contains three main parts (1) component name (2) code for performing a service (3) interfaces for accessing services.

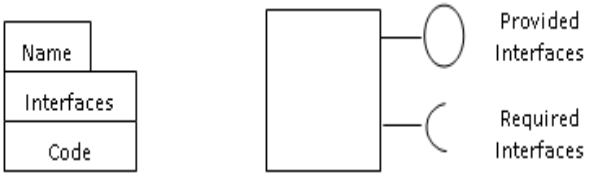

Figure 3: Software Component Model

A. *Component Composition.*

Component composition in CBD is the central issue as the components are supposed to be used as a building blocks of a software system, created from scratch or used from an existing repository and assembled together into large sub-systems or systems. Component composition is defined as

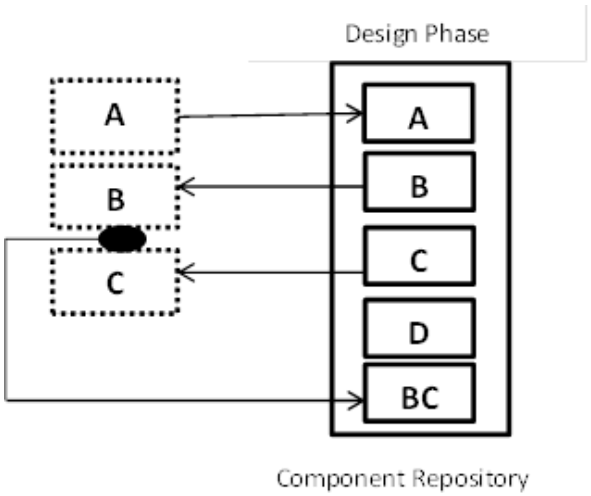

Figure 4: Design Phase Component Composition.

communication between components through their composition interfaces. In composition a software component may be composed of other components. In such a composition, the component may bind the local resources of other components and could access the resources by invoking methods on the resources. Composition can take place during different stages of the life cycle of components [20] , namely , (a) the design phase during which components are designed , defined , constructed in the source code, and possibly compiled into binaries, (b) the deployment phase , in which binaries of components are deployed into target execution environment for the system under construction (c) the runtime phase , during which the component binaries are instantiated with data and these instances are executed in the running system. Whether in the design, deployment or the runtime phase, a component composition is the interconnection of the components from an atomic component into the composition of two and successively into a cluster, sub-system and up to overall software system. In our evaluation we are concerned with the design phase so at design phase components have to be constructed, cataloged and stored in the repository [21]. The so created components can be fetched from the repository and composed into the composites and stored back in the repository. Fig. 4 above show the design phase composition

B. *Various Component Model Found And Their Composition***.**

Software components may be available in many different forms ranging from procedure and object libraries up to stand alone applications. A software component may be already composed of other components. There are various software component models. In [21] the component models are grouped according to the component semantic and syntax or composition. Four such categories that cover the 13 models are presented. All these models are based on the Acme like ADLs. In such Acme a component is an architectural unit that represents a primary computational elements and data store of a system [22, 23]. A set of ports defines the interface of the component. These ports either act as sink to receive or source to send the data between components. Below fig.5 shows the components and the composition of Acme component Model.

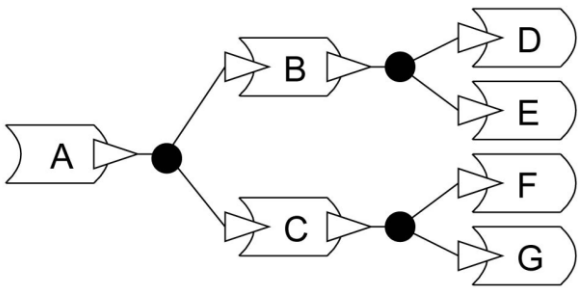

Figure 5: Component Composition of Acme Component Model

Other models like COM, EJB, .NET[24], CCM [25]. All these follow the representation like EJB model. In

EJB a component is an enterprise bean class, hosted and managed by an EJB container on J2EE server [26]. The EJB container is responsible for the enterprise beans execution, transaction management, remote connectivity and java naming and Directory Interface (JNDI) lookups. The java class for the enterpriser bean defines the methods of bean. EJB container uses to two types of interfaces Home interface and Remote interface to manage and run the beans. These interfaces are the gateways for the component for client application. The home interfaces are responsible for the life-cycle of the component such as the methods like create, locate, and destroy. The Remote interface has the methods that relate to a particular bean instance (i.e. task performed by a bean). There are three types of EJB Entity EJB, session EJB, and Message driven EJB [26].

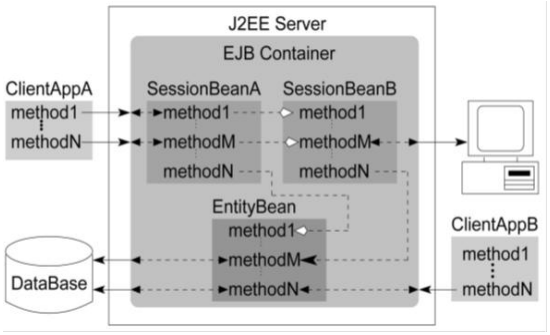

Figure 6: EJB Component Composition

A session bean represents transient user-specific data that will die when user disconnects, A message-driven bean is an Entity EJB represents the persistent global data from the data base, enterprise bean that allows J2EE applications to process messages asynchronously. Fig.6 below shows the composition of an EJB.

Other component models like Kola [27], [28], SOFA[29] and KorbaA[30], follow the representation and model of Kola. In Kola the component having two parts the component specification and implementation [31]. These components are defined in an ADL like language, in which IDL for defining the interfaces of the component, CDL for defining component and DDL for specifying local data. Below fig.7shows the composition of Kola components.

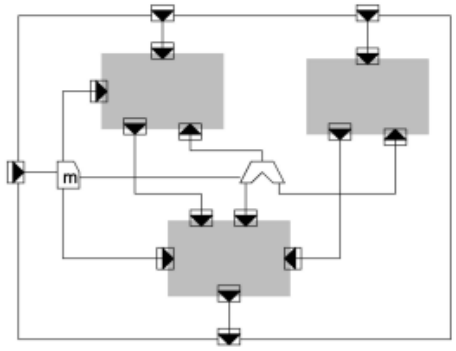

Figure 7: Kola Component composition

The dark square represents the component and the little squares with triangle like structure on the edges represents the interfaces of the component whose direction depicts the direction of function call.

V. SECURITY EVALUATION FRAMEWORK

In Component Based Design (CBD), various component models exists today with varying terminology but in general a component is a unit of functionality with well-defined ports or interfaces through which its functionality is exposed. In the current software development component based design play a vital role in the process of software development. In CBD a system is the interconnection or the composition of these components to provide the required functionality. These components may be either locally configured or remote. A system goes from several phases from its design and development phases to the running environment. The evaluation of system for security at the early architectural and design phases considerably reduces the cost and efforts at further stages. Beside that these early metrics act as an indicator for both the system developers and consumers to explicitly know the level of security of the system based on the facts at hand. The architecture of the system considerably effects on the security of the overall system. In CBD the architecture of a system defines how the overall system can be composed. In our evaluation framework we evaluate the system based on the following and derive the security metrics for the fundamental and most important security attributes Availability, Confidentiality and Integrity mentioned in section 2 above.

- Component Composition and Dependencies
- Inter Component information/data flow and resource Sharing.

A. Component Composition and Dependencies.

As stated earlier in CBD, a system is composed of various components each having the unit of functionality with provided and required interfaces. So a component may be either composite of other components which in turn may also a composite of other and finally leads to atomic components. An atomic component is one which is atomic in nature and provides the functionality without the calling upon the services of the other component. Different composition models are there but each model focus on the behavior together with one or more presentations.

In our evaluation frame work we have used the UML component composition as shown in fig 8. For simplicity we have followed a tree like organization of the system starting from the root (composites) to the leaves (atomic) components. As shown in diagram in UML a component presents its behavior by one or more required and provided interfaces (ports). Any

required service is represented by a socket and every provided service is represented by a lollipop [32].

Such a configuration or composition can certainly affect the security level of the overall system (The attributes specified above in the section.2). Here we measure the two attributes of the software security dependency and availability based on the component composition and dependency. In CBD components provide system functionalities by interacting, cooperating and coordination [33]. Interaction, cooperation and coordination will produce dependency among them .Usually, a group of components depend on each other in order to supply a complex system functionality .Various types of dependencies exist in CBD [34]. (1) Data

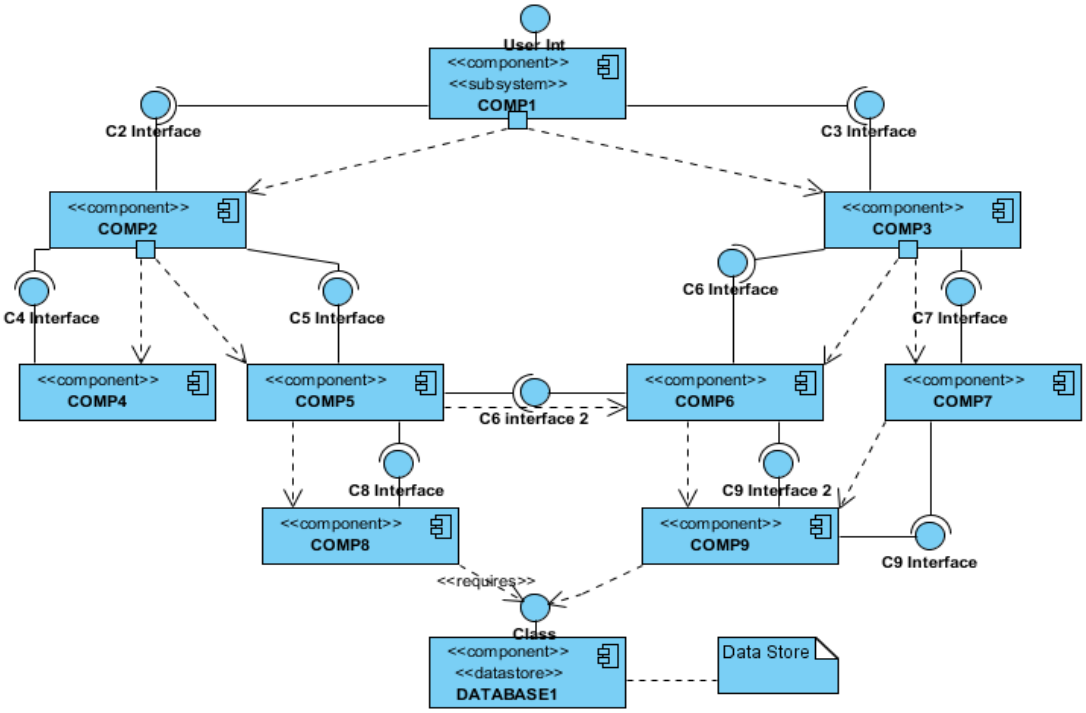

Figure 8: Software Component Composition and Dependencies

dependency produces by control integration in CBD. (3) Interface dependency is produced by user-interface integration. Usually, the interface-event dependency is the main dependency form in CBD. When one component needs another component to do something, it sends a message to trigger an event through its interface, which activates another component to response the message. (4) Time dependency represents that the behavior of one component precedes or follows the behavior of another component.

Similarly there are other dependencies like state dependency, cause and effect dependency, input output dependency and context dependency. In this section we measure the availability attribute specified in section 2 based on the dependency of the components. In fig. 8. above the dashed line represents the dependency explicitly beside the provided and required interface dependencies.

A component can exhibit two types of dependencies, *in-dependency* and *out-dependency*. The *in-dependency* represents that other components directly or indirectly dependent on it and *out-dependency* represents that component depends on the other components in the composition. We argue that the degree of components in and out dependencies can effect on the availability of the system and it becomes more critical if the components composition is remote, due to the overhead of transmission delay and of the process of remote procedure call (RPC). We compute the direct dependencies for the composition of above fig.8 using the dependency matrix [34]. The direct dependency matrix is an adjacency matrix (AM) in which each component is represented by a column and a row. If a component *Ci* is dependent on another component *Cj*, then *AM [i,j]= 1*. In general, the values of all the elements in $AM_{n*n} = (dij)_{n*n}$ as follows:

$$dij = \begin{cases} 1, & if\ ci \rightarrow cj \\ 0, & otherwise \end{cases}$$

The direct dependency matrix for the component composition of above scenario of fig.8 is as:

$$DM = \begin{array}{c} \\ c1 \\ c2 \\ c3 \\ c4 \\ c5 \\ c6 \\ c7 \\ c8 \\ c9 \\ c10 \end{array} \begin{array}{c} \begin{array}{cccccccccc} c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 & c9 & c10 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

We further use the Warshall's algorithm [35] to compute all the indirect dependencies of the scenario in fig.8 matrix FDB (full dependency metric) shows all the direct and indirect dependencies.

FBD=

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
c1	0	1	1	1	1	1	1	1	1	1
c2	0	0	0	1	1	1	0	1	1	1
c3	0	0	0	0	0	1	1	0	1	1
c4	0	0	0	0	0	0	0	0	0	0
c5	0	0	0	0	0	1	0	1	1	1
c6	0	0	0	0	0	0	0	0	1	1
c7	0	0	0	0	0	0	0	0	1	1
c8	0	0	0	0	0	0	0	0	0	1
c9	0	0	0	0	0	0	0	0	0	1
c10	0	0	0	0	0	0	0	0	0	0

We define the degree of a component dependency *deg(Ci)* as the number of components dependent on it. As mentioned earlier a component may have either *in-dependency, out-dependency* or both. From the above dependency matrix representation the degree of *out-dependency* of a component $degOUT(Ci)$ is equal to the number of 1's in row *i* of the corresponding component and the degree of *in-dependency* of a component $degIN(Ci)$ can be calculated as the number of 1's in the column *j* which can be computed as:

$$degOUT(C_i) = \sum_{j=1}^{n} (FBD_{ij})$$

And

$$degIN(C_i) = \sum_{j=1}^{n} (FBD_{ji})$$

Software architectures are now shifting from simple standalone application to large distributed software based on OSI or J2EE n-tiers, which makes the security a great challenge over the insecure cyber space. So the structure of CBD must be analyzed and evaluated for security as early at the design phase as possible. Software developers need to ensure that service should remain available in a timely manner. Availability in CBD depends up on many factors in distributed CBD. Components normally use Remote Procedure Call (RPC) to invoke and get the services provided by other components. When components call up on the other components for service, there is certainly chance of delayed response due to the chain of the dependencies among the components, marshaling and unmarshalling in case of remote procedure call (RPC), processing delay and the transmission delay. In this framework the processing and RPC delay together denoted together by P and the transmission overhead denoted by T. Also a component can act as a hub for handling the requests of other components and calling on their behalf the other components. In this section we derive the Availability metrics of a CBD based upon their dependencies (in and out) and the transmission delay.

There is a 1-N relationship between the $degIN(Ci)$ and $degOUT(Ci)$ of a component Ci i.e. for each of the component in $degIN(Ci)$, Ci may call some or all of the other components on its behalf in $degOUT(Ci)$. So the availability of a component Ci denoted by $A(Ci)$ is inversely proportional to $degIN(Ci)$ and $degOUT(Ci)$. Thus

$$A(Ci) = \frac{1}{DegIN(Ci) + \sum_{j=1}^{DegIN(Ci)} DegOUT(Ci) + \sum_{j=0}^{DegOUT(Ci)} Pj + Tj}$$

Where

$DegIN(Ci)$ is the in-dependency of the component Ci

$DegOUT(Ci)$ is the number of components Ci depends upon.

P and T represents the processing and transmission delay of the component Ci and the processing and transmission delay of all the components in the $degOUT(Ci)$ on which Ci depends for providing the required service to calling component.

The above equation can be simplified as where Po is the processing delay of the component Ci.

$$A(Ci) = \frac{1}{n + \sum_{i=1}^{n} m + T + \sum_{j=1}^{m} Pj}$$

Where

n is the $degIN(Ci)$ is the in-dependency of component Ci .

m is the $degOUT(Ci)$ is the out-dependency of component Ci.

T is the total transmission delay of invoked component Ci and the transmission delay of all the

components invoked by Ci in the $degOUT(Ci)$ for each of the component n.

With the above metric the critical component in the system can be identified at the early stages of the system development. The availability of overall system can be derived as:

$$A(CBS) = \frac{\sum_{i=1}^{N} A(Ci)}{N}$$

Where N is the number of total components in the system.

The results provided by above equation are within the range (0, 1). Lower the values on scale more the effect on the availability of the component/system.

B. Inter-Component Information/data Resource Sharing.

In CBD the coordination, interaction, cooperation and coordination makes the flow of data and information between the components. Such flow of information can affect the *confidentiality* and *integrity* of data and data storage resources among the components especially when the composition involved some third party components. The flow of information takes place through component interfaces (ports). There are two types of data flow as shown fig.9 (1) Inter component flow and (2) Intra component flow [36]. In former the flow of data/information takes place through the components interfaces by In-flow and Out-flow. The in-flow carries information from client to a provider interface through a list of in parameters and the out-flow carries information from a provider interface to a client through the list of out parameters. In later it is assumed that a data structure is used to store and retrieve information needed by the provider interface. The in-flow gets mapped to intra-component write-flow and out-flows to intra-component read-flow. The data/information flow among components can be either direct or indirect. In direct flow a component A passes the data/information directly by calling upon the methods of the component and in indirect mode a component say A passes data/information to component B which intern may passes it to other components and finally to component C. Using UML sequence diagrams it is easy to analyze the complex data/information flow among the components. In this section we derive the confidentiality and integrity metrics for CBSE based on the data/information flow and storage among the components. The main focus is to analyze the effect of each of the component in of overall system's composition on the confidentiality and integrity and to identify the most critical components having high impact on these security attributes.

Each component in CBD poses certain number of interfaces (both provided and required) for reading from and writing into the component. We believe that as the number of other system components writing into and reading from a component Ci increases, the confidentiality of that component and of the components depending on it will be affected. Also the

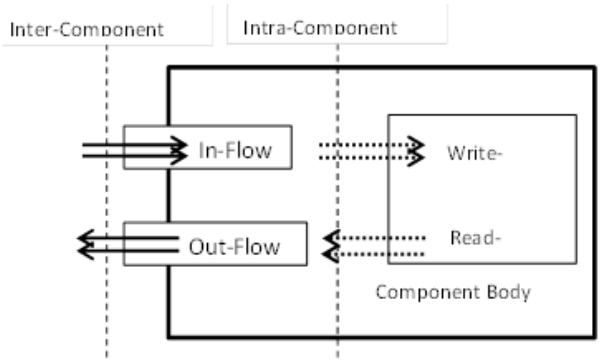

Figure 9: Component Information Flow

confidentiality is likely to be affected more by number of read operation on Ci for each of the write operation on Ci. We put it into the mathematical formulation by taking into account the assumptions as:

For each of the required interface of a component Ci the number of components writing is m.

Where $m \in degOUT(Ci)$.

Similarly for each of the provided interface of Ci the number of components reading from it are n.

Where $n \in degIN(Ci)$.

The confidentiality metric of the component Ci is as:

$$COD(Ci) = \frac{1}{n^2 m(Iw\ Ir) + 1}$$

Where

Iw is the total number of writing interfaces.

Ir is the total number of reading interfaces and, $Iw + Ir = I$.

We argue that as the number of component reading from Ci i.e.as the n increase for each of the component in m performing a write operation, the confidentiality of Ci is likely to be affected more than the increase in number of Write i.e. m alone. With the above metric

the system developer can easily identify the most critical component in the system by comparing the results and be able to modify the initial design decisions. The confidentiality level of overall system can be computed by following equation.

$$COD(S) = \frac{\sum_{i=1}^{N} COD(Ci)}{N}$$

Where, N is the total number of component in the system composition?

Similarly we compute and derive the metric for integrity of each of the component in the system composition and for the overall system. Integrity gets affected when an unauthorized and uncontrolled change to data/information is made. As the complexity of the component composition increases, it becomes very difficult to keeping track and ensuring the integrity of the system. In our evaluation we have considered, as the number n of component capable of sending the data/information to another component say Ci increases the integrity of that component Ci gets affected. The goal is to identify the most critical components in the system composition by which the integrity of the system is likely to be affected more. These early indicators provide the decision making capabilities to designer for taking the necessary actions and if required. In this context we define that the integrity of a component Ci is inversely proportional to the number of components that can send data/information into Ci through interfaces Iw. So the integrity of a component can be represented by the following mathematical equation.

$$INT(Ci) = \frac{1}{Iw * degOUT(Ci)}$$

Where Iw the number is required interfaces of Ci i.e. the write interfaces of Ci.

The overall integrity of the system formulated as :

$$INT(S) = \frac{\sum_{i=1}^{N} INT(Ci)}{N}$$

Where ***N*** is the total number of components in the system composition.

To keep the results on similar scale, the above equation is further simplified as.

$$INT(S) = \frac{1}{N * \sum_{i=1}^{N} degOUT(Ci) * Iw}$$

As in case of Availability, here also the range of output values is within the range (0, 1) and lower the resultant values the more effect on the component/system will be.

VI. CONCLUSION AND FUTURE SCOPE

In this paper we have analyzed the various software architectures and there place in software development. We have also analyzed the non-functional security requirements specification centric to the information security attributes. Our focus was on to the component based software development (CBD), because of the level of abstraction and its place in the software development which makes it the best level of software architectural abstraction for evaluating the quality attributes. We have also analyzed the various component models currently in use. Finally we have proposed a framework for the security evaluation of component based software based on component dependencies and information flow and derived the security metrics for the fundamental three attributes, Confidentiality, Integrity and Availability of security. The proposed metrics can act as the early indicator of the security for the system developers. In this we have not provided the empirical evaluation of the proposed framework, which will be the future enhancement.

REFERENCES

[1]. D. P. Gilliam, T. L. Wolfe, J. S. Sherif, and M. Bishop. “Software security checklist for the software life cycle.” In Proceedings of the Twelfth IEEE International Workshop on Enabling Technologies: Infrastructure for Colaborative Enterprises (WETICE’03), 2003.

[2]. W. Jansen, “Directions in security metrics research”, U.S. National Institute of Standards and Technology, NISTIR 7564, Apr. 2009

[3]. M. Howard and S. Lipner. The Security Development Lifecycle. Microsoft Press, 2006.

[4]. Oman, P., Risley, A., Roberts, J., and Schweitzer III, E.O. “Attack and Defend Tools for Remotely Accessible Control and Protection Equipment in Electric Power Systems,” 55th Annual Conference for Protective Relay Enginers, Texas A&M University, April 9–11, 2002, College Station, TX. <http://www.selinc.com/techpprs/6132.pdf> (4 Mar. 2003)

[5]. Bayuk J.L. "Alternate security metrics" Eight International conference on Information Technology: New Generation IEEE, 2011.
[6]. S. B. Lipner. The Trustworthy Computing security development Life Cycle. In Proceedings of 20th Annual Computer Security Applications Conference. IEEE Computer Society, December 2004, pp. 2-13.
[7]. D. Firesmith "Specifying reuse able security requirements" Journal of object technology vol.3, No. 1, Jan 2004. Pp. 61-75.
[8]. R. Savola, "Requirement Centric Security Evaluation of Software Intensive Systems," DepCOSRELCOMEX '07, Szklarska Poreba, Poland, jun., 14-16,2007, pp.135-142
[9]. B Thuraisingham "Challenges and Future Directions of Software Technology: Secure Software Development" 34th Annual IEEE Conference on Computer Software and Application, 2010.
[10]. Hong Mei, Jichuan Chang, Fuqing Yang, "Composing Software Components at Architectural Level", IFIP WCC2000, Beijing, 2000.8
[11]. Perry, D.E, Wolf, A.L, Foundations for the study of software architecture, ACM SIGSOFT Software engineering notes,1992, 17(4), 40-52
[12]. IEEE 1471:2000—Recommended practice for architectural description of software intensive systems. Los Alamitos, CA: IEEE. 2000.
[13]. Bohem, B and W.L. Scherlis, "Megaprogramming." In Proceedings of the DARPA Software Technology Conference 1992, Los Angeles, CA, April 28-30, (Meridien Corp., Arlington, VA) 1992. pp. 63-82.
[14]. B. Boehm and V. Basili, "Software defect reduction top 10 list," Foundations of empirical software engineering: the legacy of Victor R. Basili bach, and M. V. Zelkowitz, Eds. Heidelberg, Germany: Springer, 2005, pp. 426-431.
[15]. M.Shaw, D.Garlan Software Architecture, Prentice Hall, Englewood Cliffy, NJ, USA, 1996.
[16]. C. Szyperski. Component Software: Beyond Object-Oriented Programming. Addison-Wesley, 1998.
[17]. G. Pour, "Component-Based Software Development Approach: New Opportunities and Challenges," Proceedings Technology of Object-Oriented Languages, 1998. TOOLS 26., pp. 375-383.
[18]. A.W. Brown, S. Johnston, and K. Kelly. Large-scale, using service-oriented architecture and component-based development to build web service applications. Rational Software White Paper TP032, 2002
[19]. Kung-Kiu Lau and Zheng Wang. Software component models. IEEE Transactions on Software Engineering, 33(10), October 2007, pp. 709-724.
[20]. B. Christiansson, L. Jakobsson, and I. Crnkovic, "CBD Process," Building Reliable Component-Based Software Systems, I. Crnkovic and M. Larsson, eds., pp. 89-113, Artech House, 2002.
[21]. Kung-Kiu Lau and Zheng Wang "Software Component Models" , IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 33, NO. 10, OCTOBER 2007.
[22]. P. Clements, "A Survey of Architecture Description Languages," Proc. Eighth Int'l Workshop Software Specification and Design (IWSSD'96), pp. 16-25, 1996.
[23]. N. Medvidovic and R.N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages, "IEEE Trans. Software Eng., vol. 26, no. 1, pp. 70-93, Jan.2000.
[24]. A. Wigley, M. Sutton, R. MacLeod, R. Burbidge, and S. Wheelwright, Microsoft .NET Compact Framework (Core Reference). Microsoft Press, Jan. 2003.
[25]. G. Alonso, F. Casati, H. Kuno, and V. Machiraju, Web Services: Concepts, Architectures and Applications. Springer-Verlag, 2004.
[26]. L. DeMichiel and M. Keith, Enterprise JavaBeans, Version 3.0. SunMicrosystems, 2006.
[27]. R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala Component Model for Consumer Electronics Software," Computer, vol. 33, no. 3, pp. 78-85, Mar. 2000.
[28]. R. van Ommering, "The Koala Component Model," Building Reliable Component-Based Software Systems, I. Crnkovic and M. Larsson, eds., pp. 223-236, Artech House, 2002.
[29]. F. Pla´sil, D. Balek, and R. Janecek, "SOFA/DCUP: Architecture for Component Trading and Dynamic Updating," Proc. Fourth Int'l Conf. Configurable Distributed Systems (ICCDS '98), pp. 43-52, 1998.
[30]. C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wu¨st, and J. Zettel, Component-Based Product Line Engineering with UML. Addison-Wesley, 2001.
[31]. R. van Ommering, "The Koala Component Model," Building Reliable Component-Based Software Systems, I. Crnkovic and M. Larsson, eds., pp. 223-236, Artech House, 2002.
[32]. J. Cheesman and J. Daniels, UML Components: A Simple Process for Specifying Component-Based Software. Addison-Wesley, 2000.
[33]. Binbin Qu, Zuwen Chen, Yansheng Lu "An approach of test sequence generation for component-based software" 2nd International Conference on Future Computer and communication (ICFCC) vol.2. pp. 370-373, May 2010.

[34]. B. Li, "Managing Dependencies in Component-Based Systems Based on Matrix Model" Proc. of Net.ObjectDays Conf., pp.22-25, 2003.
[35]. Rosen, Kenneth H., Discrete Mathematics and its Applications, Third Edition, McGraw-Hill, Inc, 1994.
[36]. M. Abdellatief, "Component-Based Software System Depencey Metrics based on Component Information Flow Measurement", The Sixth International Conference on Software Engineering Advances, ISBN: 978-1-61208-165-6 ICSEA 2011.1
[37]. W. Jansen, "Directions in security metrics research", U.S. National Institute of Standards and Technology, NISTIR 7564, Apr. 2009, 21
[38]. M. Howard and S. Lipner. The Security Development Lifecycle. Microsoft Press, 2006.

Irshad Ahmad Mir is currently pursuing Ph.D degree program in computer science department at University of Kashmir, India.He did his bachelor's degree in computer application from Amar Singh College Srinagar India and Master Degree in computer application from Kashmir University India.
E-mail: irshad.mir@hotmail.com.

Dr. SMK Quadri is Head, PG department of computer sciences, Kashmir University, India. He did his Mtech in computer application from Indian school of Mines and Ph.D in computer sciences from Kashmir University, India.

